- IPMI -

IPMI v1.5 Addenda, Errata, and Clarifications

Intelligent Platform Management Interface Specification
v1.5, revision 1.1

Addendum Document Revision 4

9/26/03

Contents

Introduction

This document presents cumulative errata and clarifications applying to the *Intelligent Platform Management Interface Specification v1.5*, revision 1.1. Section, table, and figure references are given relative to that revision of the specification, unless otherwise noted. Where examples are given, text additions are shown with double underlines, and text deletions are shown with strike-through. Also included are errata against the *IPMB v1.0 Address Allocation, revision 1.1* specification and *ICMB Specification Version 1.0, revision 1.2*.

As of this writing, revision 1.0 of the *Intelligent Platform Management Interface Specification v1.5, revision 1.1* and the other IPMI specifications are available from the IPMI Web Site at:

http://developer.intel.com/design/servers/ipmi

Errata Numbers

The errata numbers pick up from where the numbers for IPMI v1.5 revision 1.0 left off. This is done to help avoid confusion when referring to errata across revisions of the specification. Some errata numbers are skipped in this document. This is intentional. The errata numbers are derived from numbers used for tracking errata and clarification requests within the IPMI Promoters group. The gaps in the sequence result from requests that have been dropped or that are still in progress.

10/30/02 Addenda, Errata, and Clarifications

E256 Addendum - Timestamp Synch Event

A new offset is added to the System Event sensor to indicate that a timestamp synch event has occurred. This event can be used to determine the relationship between relative-offset timestamped events, and those with non-relative timestamps.

Table 36-3, Sensor Type Codes

...

System Event	12h	00h … 05h	… Timestamp Clock Synch. This event can be used to record when changes are made to the timestamp clock(s) so that relative time differences between SEL entries can be determined. See note [1]. Event Data 2 [7] - first/second 0b = event is first of pair. 1b = event is second of pair. [6:4] - reserved [3:0] - Timestamp Clock Type 0h = SEL Timestamp Clock updated. (Also used when both SEL and SDR Timestamp clocks are linked together.) 1h = SDR Timestamp Clock updated.

...

1. To track the relationship between timestamps, the timestamp change events should be logged in pairs - the first event being logged just before the timestamp clock update followed by a second event that is logged after the timestamp clock has been updated. This enables software that reads the SEL to be able to determine time relationship between events that were logged before the update and those logged afterward. The generation of these events is normally the responsibility of the software that changes the timestamp clock. Note that some implementations may queue events prior to their being logged. It is recommended that generic software read the SEL to verify that the first event has been recorded with the relative timestamp before setting the new timestamp value and generating the second event.

E257 Addendum - Table 36-3, Sensor Type Codes

LAN Leash Lost (system has been unplugged from LAN) has no intrinsic data to identify which network controller the leash was lost on. Suggest adding optional Event data 2 as a byte to identify which network controller that LAN Leash state was detected on. As a convention, 0 should typically correspond to the first network controller, and be the value used when there's only one network controller for which LAN Leash Lost is detected by system management.

Table 36-3, Sensor Type Codes

…

Physical Security (Chassis Intrusion)	05h	00h 01h 02h 03h 04h 05h 06h	General Chassis Intrusion Drive Bay intrusion I/O Card area intrusion Processor area intrusion LAN Leash Lost (system is unplugged from LAN) *The Event Data 2 field can be used to identify which network controller the leash was lost on where 00h corresponds to the first (or only) network controller.* Unauthorized dock/undock FAN area intrusion (supports detection of hot plug fan tampering)

E258 Addendum - FRU Specification

The IPMI *Platform Management FRU Information Storage Definition v1.0* specification did not call out a value to use if the Mfg. Date / Time was unspecified. A value has been defined for 'unspecified' for mfg date/time, as follows:

3	**Mfg. Date / Time** Number of minutes from 0:00 hrs 1/1/96. LSbyte first (little endian) 00_00_00h = unspecified.

E259 Typo - Section 17.9, Broadcast 'Get Device ID'

The text has been updated to clarify that broadcast commands such as *Broadcast Get Device ID* are not bridged, but can be delivered to IPMB or PCI SMBus using the Master Write-Read command.

17.9 Broadcast 'Get Device ID'

This is a broadcast version of the 'Get Device ID' command that is provided for the 'discovery' of Intelligent Devices on the IPMB. It is only specified for use on the IPMB. Discovery of management controllers on a PCI Management Bus is handled via the SMBus 2.0 'ARP' protocol. See [SMBUS] for more information.

The Broadcast 'Get Device ID' command is not bridged but can be delivered to the IPMB using *Master Write-Read* commands.

E260 Clarification - Section 6.11.1, Session-less Connections

Add notes to indicate that commands that are executable 'outside' of a session can be executed within a session, but would need valid session IDs, etc., to be accepted.

6.11.1 Session-less Connections

A session-less connection is unauthenticated. There is no 'user login' required for performing IPMI messaging. The System Interface and IPMB are examples of session-less connections.

A special case of a session-less connection can occur over an interface that supports session-based messaging. Session-based connections have certain commands that are accepted and responded to "outside of a session". When that occurs, the channel is effectively operating in a session-less manner for those commands. Commands that are handled outside of a session have fixed values for session-specific fields in the message. For example, when the "Get Channel Authentication Capabilities" is sent over a LAN channel outside of a session, it is sent with the session ID set to NULL and authentication type set to NONE in the IPMI session header of the message. Note that commands that are accepted "outside of a session" can also be accepted within the context of a session, in which case they must have valid session IDs, etc., in the session header in order to be accepted.

E261 Addendum - Table 18-20, Get Session Info Command

Some implementations may not be able to return LAN information due to hardware limitations. In order to support such implementations, the table description of response bytes 8:19 for Channel Type = 802.3 LAN have been made optional, as follows. A typo of UPD → UDP was also corrected.

Table 18-20, Get Session Info Command

...

	The following bytes 8:18 are optionally returned if Channel Type = 802.3 LAN:
8:11	IP Address of remote console (MS-byte first). Address that was received in the *Activate Session* command that activated the session.
12:17	MAC Address (MS-byte first). Address that was received in the *Activate Session* command that activated the session.
18:19	Port Number of remote console (LS-byte first). Port Number that was received in UDP~~UPD~~ packet that held the *Activate Session* command that activated the session.

...

E262 Clarification - Section 35.2, Event/Reading Type Code

The non-digital (more than 2 valued) discrete offsets in table 36-2 describe transitions from one state to another. When these offsets are returned as sensor readings, they represent the current and the previous state of the sensor. If the BMC cannot determine the previous state of the sensor, or an appropriate offset is not provided, it may return any transition that shows the current state.

Added following text to section 35.2:

When event/reading offset values are returned as sensor readings (using the *Get Sensor Reading* command) they represent the present state of the sensor. For example, for the offset that is described as "Transition to Idle" the present state for the asserted condition of the offset would be interpreted as "Idle". In some cases, the event/reading code also includes information that implies the previous state from which the present state was entered. For example, the generic event/reading type code for the "Severity Event States" includes offsets such as "Transition to Non-critical from OK". However, when the platform management subsystem is initialized, the BMC may not know what the previous state was. In the case that the BMC cannot determine the previous state of the sensor, or an appropriate offset is not provided, it can return any offset that shows the correct current state.

By convention, if the previous state is unknown and there is a choice of possible offsets for the present state most implementations will return the offset that corresponds to a previous state that is 'OK'. For example, it would return the offset for "Non-critical from OK" rather than "Non-critical from more severe". This is not a requirement, however. Therefore, since the actual previous state may be unknown, it is recommended that management software ignores the 'directionality' information inferred by the offset for the present reading. For example, when interpreting the present reading for the "Severity Event Status" it should present the present state just as "Non-critical" and ignore the 'from OK / from more severe' aspect associated with the offset.

If software cares about the transition direction associated with entering a given present state, it should get that information from the sensor event status (using the *Get Sensor Event Status* command). If a sensor returns '0b' values for both assertion and deassertion offsets in the *Get Sensor Event Status* command, it means that the sensor has been initialized with the previous state unknown. I.e. there have been no detected state transitions yet that would cause the event status bits to become set. In this case, it is clear that there is no 'directionality' information available and software should generally ignore any 'directionality' that is implied by the present reading.

E263 Clarification - Section 34.2, Software detection of Entities

The section did not discuss how software should handle contained entities when the presence sensor for the container entity indicates the container entity is absent. The following is added to the bullet list:

- For a container entity that has a presence sensor associated with it, if the presence sensor indicates the container entity is absent, software should consider the contained entities and associated sensors as also being absent. Note that some software may not interpret Entity-association records. Therefore, if a given sensor is described in the SDRs and remains accessible when the container entity is absent, either the sensor should return 'scanning disabled' or, if the sensor has a presence bit, should return that the monitored entity is 'not present'.

E264 Typo - Table 36-1, Event/Reading Type Code Ranges

In Table 36-1 Event/Reading Type Code Range, 'Generic' Event/Reading Type Code Range was listed as 02h-0Bh. But in table 36-2, 'ACPI Device power state' is specified as type code 0Ch, so the range in table 36-1 is changed to 02h-0Ch, as follows:

Table 36-1, Event/Reading Type Code Ranges

...

Generic	02h-0Ch~~0Bh~~	discrete	Generic Discrete. Indicates a sensor that utilizes an Event/Reading Type code & State bit positions / event offsets from one of the sets specified for Discrete or 'digital' Discrete Event/Reading class in *Table 36-2, Generic Event/Reading Type Codes*, below.

E265 Typo - Section 6.11.14, Additional Session Specifications and Characteristics

Section 6.11.14, the second bullet from the last one had an incomplete last sentence. This sentence was to be an example of how the commands could be used to a given managed system. The bullet item has been completed as follows:

- An Operator can optionally use the *Get Channel Info* and *Get Session Info* commands to retrieve the address of parties with open sessions and their present privilege level. This is to allow a remote console to coordinate with another remote console that already has an active session. This can be used to allow software to coordinate access to the system. For example, if management software running at Console "A" wished to remotely reset a given system, it could first see whether another console had an active session with the system to be reset. It could then use information from the *Get Channel Info* and *Get Session Info* commands to send a message directly to the other console, notifying it of the pending reset.

E266 Addendum - Table 36-3, Sensor Type Codes

A new sensor type for "Session Audit" has been added to the specification. This sensor type provides a mechanism for logging session activation and deactivation and the associated cause and User ID at the time.

Table 36-3, Sensor Type Codes

...

Session Audit	2Ah	00h 01h	Session Activated Session Deactivated *The Event Data 2 & 3 fields can be used to provide an event extension code, with the following definition:* Event Data 2 7:6 reserved 5:0 User ID for user that activated session. 00_0000b = unspecified. Event Data 3 7:6 reserved 5:4 Deactivation cause 00b = Session deactivatation cause unspecified. This value is also used for Session Activated events. 01b = Session deactivated by C*lose Session* command 10b = Session deactivated by timeout 11b = Session deactivated by configuration change 3:0 Channel number that session was activated/deactivated over. Use channel number that session was activated over if a session was closed for an unspecified reason, a timeout, or a configuration change.

...

E270 Errata - Table 22-12, Boot Option Parameters

The "Service Partition discovered" bit was described as being cleared after system resets and power ups, unless a scan was requested. This complicates the BIOS/BMC interaction, and also meant that in most cases a remote console application that wished to restart a managed system would not have information whether or not a service partition present before initiating the restart. Therefore, in Table 22-12 parameter 2, service partition scan, is changed from "semi-volatile" to "non-volatile" and bit 0 is specified to retain the value from the last scan.

...

service partition scan (non-volatile)	2	data 1 [7:2] - reserved [1] - 1b = Request BIOS to scan for specified service partition. BIOS clears this bit after the requested scan has been performed. [0] - 1b = Service Partition discovered. BIOS sets this bit to indicate it has discovered the specified service partition. The bit retains the value from the last scan. Therefore, to get up-to-date status of the discovery state, a scan may need to be requested.

...

E271 Addendum - Identify LED control for Terminal Mode

Add an optional Terminal Mode command to allow Identify LED to be turned on.

The following Terminal Mode text command is defined to allow the BMC to signal a system's location by, for example, blinking an LED or emitting a beep.

Table 13-13, Terminal Mode Text Commands

...

Command Text	Description
SYS IDENTIFY	Causes the BMC to signal the system's location (e.g. with a blinking led or beep). This is intended to locate the system amongst a rack of systems. The BMC will signal the system's location for 15 seconds and then stop signaling. This is a text version of the optional *Chassis Identify* command.
SYS IDENTIFY –ON <XX>	Causes the BMC to signal the system's location (e.g. with a blinking led or beep) for a specific amount of time. XX is an optional hex-ASCII byte representing the number of seconds the BMC is to cause the system to identify itself. If XX is not supplied, the BMC will signal the system's location for 15 seconds and then stop signaling. This is a text version of the optional *Chassis Identify* command.
SYS IDENTIFY –OFF	Causes the BMC to stop signaling the system's location. This has no effect if the system is not currently identifying itself. This is a text version of the optional *Chassis Identify* command.

...

E272 Typo and Clarification - Section 18.15.1, AuthCode Algorithms

The second sentence of the section was incomplete. It has been deleted and the first sentence clarified as follows:

> The following table lists the AuthCode calculation mechanism and field usage for the *Activate Session* command, authenticated packets, and the *Get AuthCode* command.

E273 Typo - Table 5-1, Network Function Codes

The name 'Compact PCI' for a defining body is incorrect. The name of the defining body is PICMG. This is corrected in the table as follows:

Table 5-1, Network Function Codes

...

2Ch-2Dh	Group Extension	Non-IPMI group Requests and Responses	The first data byte position in requests and responses under this network function identifies the defining body that specifies command functionality. Software assumes that the command and completion code field positions will hold command and completion code values. The following values are used to identify the defining body: 00h** PICMG - PCI Industrial Computer Manufacturer's Group. (www.picmg.com) 01h DMTF Pre-OS Working Group ASF Specification (www.dmtf.org) all other Reserved When this network function is used, the ID for the defining body occupies the first data byte in a request, and the second data byte (following the completion code) in a response.

E274 Addendum - Table 36-3, Sensor Type Codes

Added a value, 6, for slot type #6 for "AdvancedTCA" in the Event Data 2 byte of the slot/connector sensor (21h) in Table 36-3, Sensor Type Codes.

E275 Addendum - Address Allocation document

Request to remove 50h & 6Eh as Access.bus addresses to clear address space, because Access.bus is no longer used. Incorporated these addresses in adjacent range. This change simplifies support for IPMI in AdvancedTCA™-based systems.

Class KEY:

- - Reserved for I^2C ~~& Access.bus~~ specification functions.
- B Reserved for Board Set manufacturer use.
- I Defined by IPMI Group for Intelligent Platform Management Bus use.
- c chassis. Reserved for use by system integrator for chassis-specific functions. Not intended for board set or baseboard module.
- a For third-party add-ins. Note: add-ins should only use intelligent controllers capable of being configured to at least 8 different addresses in the 'a' range.

IPMB Address Allocation

Addr	Use	Typical Device	Addr	Use	Typical Device	Addr	Use	Typical Device
00h	I	I^2C, IPMB Broadcast	50h-6Eh	c	58h, 5Ah, 5Ch = Heceta	A0h	B	SEEPROM
~~01h~~	~~-~~	~~I^2C~~	70h	B	8574A	A2h	B	SEEPROM
~~02h~~	~~-~~	~~I^2C~~	72h	B	8574A	A4h[2]	c	SEEPROM
~~04-0Eh~~	~~-~~	~~I^2C~~	74h[1]	c	8574A	A6h[2]	c	SEEPROM
10h	a	SMBus Host	76h[1]	c	8574A	A8h	B	SEEPROM
12h-16h	a		78h[1]	c	8574A	AAh	B	SEEPROM
18h	a	SMBus Alert Response Address	7Ah[1]	c	8574A	ACh	c	SEEPROM
1Ah-1Eh	a		7Ch[1]	c	8574A	AEh	c	SEEPROM
20h	I	IPMB uC (BMC)*	7Eh[1]	c	8574A	B0h-BEh	a	power supply monitoring
22h	B	uC (FPC, ICMB)*	80h-8Eh	B		C0h	B	
24h	B	uC (PBC)*	90h	B	LM75, DS1624, DS1621, 8591	C2h	B	SMBus 2.0 Device Default Address
26h	B		92h	B	LM75, DS1624, DS1621, 8591	C4h-CEh	B	
28h	B	SM Card*	94h[2]	B	LM75, DS1624, DS1621	D0h-DEh	a	
2A-2Eh	B		96h[2]	B	LM75, DS1624, DS1621	E0h-EEh	B	
30h-3Eh	a		98h[2]	c	LM75, DS1624, DS1621	~~F0h-F6h~~	~~-~~	~~I^2C~~
40h	B	8574	9Ah[2]	c	LM75, DS1624, DS1621	~~F8h-FEh~~	~~-~~	~~I^2C~~
42h	B	8574	9Ch	c	uC (pri. HSC), DS1624, DS1621			
44h	c	8574	9Eh	c	uC (prisec. HSC), DS1624, DS1621			
46h	c	8574						
48h	c	8574						
4Ah	c	8574						
4Ch	c	8574						
4Eh	c	8574						

E278 Errata and Clarifications - ICMB Specification Version 1.0, revision 1.2

Clarification: In section 2.1.8, ICMB Population Discovery, the recommendation to system software for the number of retries for the *PrepareForDiscovery* message was ambiguous. The text is changed as follows:

The ICMB protocol is designed to allow for dynamic discovery of a bus segment's population. No manual identification or manual configuration is necessary. The ICMB discovery process is driven from the system management software level. The ICMB bridge just provides the basic mechanisms to support discovery of a bus segment's chassis population.

1. The management software first sends a *PrepareForDiscovery* to its local bridge. This causes the broadcast of a *PrepareForDiscovery* message over the ICMB to prepare the bus' population for a discovery cycle. This command places the bridges into an 'undiscovered' state (with respect to the sending bridge[1]) where they will respond to the *GetICMBAddress* message. (The management software should send the ~~The~~ *Prepare for Discovery* message ~~is sent several~~ four or more times to ensure that all bridges get the message.)

Errata: Table 5-1, ICMB Bridge Commands. The response parameters did not match up with the request parameters. This is corrected as follows:

Table 5-1, ICMB Bridge Commands

...

35h	GetEventReceptionState (optional)	bridge		byte 1 - completion code byte 2 - reception state 00h = disabled 01h = enabled byte 3 - eventSA byte 4 - LUN

E279 Addendum - Table 37-12, Entity ID Codes

Table 37-12, Entity ID Codes, has the following additions to better cover current and future multi-system chassis offerings (like blade servers). These definitions are in line with those made to the SMBIOS specification last year for the same purpose.

Table 37-13, Entity ID Codes

Code		Entity
...		
41	29h	Processing blade (a blade module that contains processor, memory, and I/O connections that enable it to operate as a processing entity)
42	2Ah	Connectivity switch (a blade module that provides the fabric or network connection for one or more processing blades or modules)
43	2Bh	Processor/memory module (processor and memory together on a module)
44	2Ch	I/O module (a module that contains the main elements of an I/O interface)
45	2Dh	Processor/ IO module (a module that contains the main elements of an I/O interface)

[1] In order to better handle more than one system simultaneously performing a discovery, it is *recommended*, but not required, that the bridge maintains separate discovered/undiscovered state for at least four different sources. That is, the bridge tracks the source address of the requester for the *PrepareForDiscovery* command, and handles subsequent *SetDiscovered* and *GetICMBAddress* commands based on that address. The implementation must allow for an indefinite number of different sources of the *PrepareForDiscovery* command over time. One approach to meeting this requirement would be to maintain a list that tracks the requesters that have issued the *SetDiscovered* message and use a round-robin or LRU algorithm to replace entries in the list if it gets full.

E280 Addendum - Event-Only SDR (Type 03h)

A new SDR type for providing a more efficient record for describing 'event only' sensors.

SDR Type 03h, Event-Only Record

This record provides a mechanism to associate FRU and Entity information with a physical or logical sensor that generates events, but cannot otherwise be accessed. This is typical of software-generated events, such as events generated by BIOS. Use of this record is optional. A system is allowed to have 'Event-Only' sensors without having a corresponding Event-Only Sensor Record.

While primarily used with software-generated events, it is possible for management controllers to implement 'Event-Only' sensors. Such sensors must have the following characteristics:

- The controller cannot rely on the Initialization Agent function to enable Event Generation for the sensor, with the exception that the sensor can be 'globally' enabled/disabled with the *Set Event Receiver* command.
- Since an 'Event-Only' sensor is not required to support any of the sensor access commands, software will typically ignore the sensors and will not attempt to send IPMI commands to access or enable them. Thus, the controller cannot rely on any software accesses to the sensor. Therefore, the sensor cannot require manual-rearm for event operation.
- The sensor cannot return a units-based sensor reading. This record lacks the necessary conversion factors for management software to be able to present the reading value.

If the sensor implementation does not meet the above criteria, the appropriate Type 01h or Type 02h record must be used instead.

The controller is allowed to implement IPMI sensor commands for an Event-Only sensor, such as *Get Sensor Reading*, or *Get Sensor Event Status*. Use of the Event-Only SDR implies that the sensor access commands are not explicitly supported. Therefore, software should avoid issuing sensor access commands for sensors that use the Event-Only SDR.

Table 37-~~23~~, ~~Compact~~ Event-Only Sensor Record - SDR Type 0~~2~~3h

byte	Field Name	size	Description
	SENSOR RECORD HEADER		
1:2	Record ID	2	The Record ID is used by the Sensor Data Repository device for record organization and access. It is *not* related to the sensor ID.
3	SDR Version	1	Version of the Sensor Model specification that this record is compatible with. **51h** for this specification. *BCD encoded with bits 7:4 holding ~~-~~ the Least Significant digit of the revision and bits 3:0 holding the Most Significant bits.*
4	Record Type	1	Record Type Number = ~~02h~~03h, Event-Only ~~Compact~~ Sensor Record
5	Record Length	1	Number of remaining record bytes following.
	RECORD KEY BYTES		
6	Sensor Owner ID	1	[7:1] - 7-bit I^2C Slave Address, or 7-bit system software ID[1] [0] - 0b = ID is IPMB Slave Address, 1b = system software ID

byte	Field Name	size	Description
7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number) [3:2] - FRU Inventory Device Owner LUN. LUN for Write/Read FRU commands to access FRU information. 00b otherwise. [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner.
8	Sensor Number	1	Unique number identifying the sensor behind the given slave address and LUN. Code FFh reserved.
	RECORD BODY BYTES		
9	Entity ID	1	Indicates the physical entity that the sensor is monitoring or is otherwise associated. See Table 37-12, Entity ID Codes.
10	Entity Instance	1	[7] - 0b = treat entity as a physical entity per Entity ID table 1b = treat entity as a logical container entity. For example, if this bit is set, and the Entity ID is 'Processor', the container entity would be considered to represent a logical 'Processor Group' rather than a physical processor. This bit is typically used in conjunction with an Entity Association record. [6:0] - Instance number for entity. (See section *33.1, System- and Device-relative Entity Instance Values* for more information) 00h-5Fh system-relative Entity Instance. The Entity Instance number must be unique for each different entity of the same type Entity ID in the system. 60h-7Fh device-relative Entity Instance. The Entity Instance number must only be unique relative to the management controller providing access to the Entity.
11	Sensor Type	1	Code representing the sensor type. From the Table 36-3, Sensor Type Codes. E.g. Temperature, Voltage, Processor, etc.
12	Event / Reading Type Code	1	Event/Reading Type Code. From the Table 36-1, Event/Reading Type Code Ranges.
13	Sensor Record Sharing	2	Byte 1: [7:6] - reserved ID String Instance Modifier Type [5:4] - 00b = numeric 01b = alpha Share Count [3:0] - Share count (number of sensors sharing this record). Sensor numbers sharing this record are sequential starting with the sensor number specified by the *Sensor Number* field for this record. E.g. if the starting sensor number was 10, and the share count was 3, then sensors 10, 11, and 12 would share this record. Byte 2: Entity Instance Sharing [7] - 0b = Entity Instance same for all shared records 1b = Entity Instance increments for each shared record [6:0] - ID String Instance Modifier Offset Multiple Discrete sensors can share the same sensor data record. The ID String Instance Modifier and Modifier Offset are used to modify the Sensor ID String as follows: Suppose sensor ID is "Temp " for 'Temperature Sensor', share count = 3, ID string instance modifier = numeric, instance modifier offset = 5 - then the sensors could be identified as: Temp 5, Temp 6, Temp 7 If the modifier = alpha, and offset = 26, then the sensors could be identified as: Temp AA, Temp AB, Temp AC (alpha characters are considered to be base 26 for ASCII)

byte	Field Name	size	Description
15	reserved	1	reserved. Write as 00h.
16	OEM	1	Reserved for OEM use.
17	ID String Type/Length Code	1	Sensor 'ID' String Type/Length Code, per Section 37.14, Type/Length Byte Format.
18:+N	ID String Bytes	N	Sensor ID String bytes. Only present if non-zero length in Type/Length field. *16 bytes, maximum.*

Notes:
1. 7-bit I^2C Slave Address field. By convention, the I^2C slave address is represented as an eight-bit number with the least-significant bit always 0. E.g. 20h = 00100000b. The 7-bit Slave Address field holds the most-significant 7 bits of this value. E.g. 0010000b.

E281 Addendum - Table 36-3, Sensor Type Codes, offsets for slot/connector sensor

Add offsets to better cover use of a given slot for holding a hot spare device, and for directly representing hot-swap slots for FAN and DIMM instead of requiring going through the SDR to get the Entity ID.

Table 36-3, Sensor Type Codes

...

Slot / Connector	21h	00h	Fault Status asserted
		01h	Identify Status asserted
		02h	Slot / Connector Device installed/attached [This can include dock events]
		03h	Slot / Connector Ready for Device Installation - Typically, this means that the slot power is off. The Ready for Installation, Ready for Removal, and Slot Power states can transition together, depending on the slot implementation.
		04h	Slot/Connector Ready for Device Removal
		05h	Slot Power is Off
		06h	Slot / Connector Device Removal Request - This is typically connected to a switch that becomes asserted to request removal of the device)
		07h	Interlock asserted - This is typically connected to a switch that mechanically enables/disables power to the slot, or locks the slot in the 'Ready for Installation / Ready for Removal states' - depending on the slot implementation. The asserted state indicates that the lock-out is active.
		08h	Slot is Disabled
		09h	Slot holds spare device. *The Event Data 2 & 3 fields can be used to provide an event extension code, with the following definition:* Event Data 2 7 reserved 6:0 Slot/Connector Type 0 PCI 1 Drive Array 2 External Peripheral Connector 3 Docking 4 other standard internal expansion slot 5 slot associated with entity specified by Entity ID for sensor 6 AdvancedTCA 7 DIMM/memory device 8 FAN all other = reserved Event Data 3 7:0 Slot/Connector Number

...

E282 Clarification - Table 18-20, Get Session Info Command

The command did not indicate what value should be returned for the session ID if the request was for an entry that did not correspond to an active session. This is corrected as follows:

Table 18-20, Get Session Info Command

	…	
IPMI Response Data	1	Completion Code
	2	Session Handle presently assigned to active session. FFh = reserved. Return 00h if no active session associated with given session index.
	…	

E283 Addendum - Table 36-3, Sensor Type Codes, offsets for Memory Sensor

The following additional offset values have been added to the memory sensor in Table 36-3, Sensor Type Codes to support hot-spare/hot-swap memory.

Table 36-3, Sensor Type Codes

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
			…
Memory	0Ch	00h	Correctable ECC / other correctable memory error
		01h	Uncorrectable ECC / other uncorrectable memory error
		02h	Parity
		03h	Memory Scrub Failed (stuck bit)
		04h	Memory Device Disabled
		05h	Correctable ECC / other correctable memory error logging limit reached
		06h	Presence detected. Indicates presence of entity associated with the sensor. Typically the entity will be a 'memory module' or other entity representing a physically replaceable unit of memory.
		07h	Configuration error. Indicates a memory configuration error for the entity associated with the sensor. This can include when a given implementation of the entity is not supported by the system (e.g., when the particular size of the memory module is unsupported) or that the entity is part of an unsupported memory configuration (e.g. the configuration is not supported because the memory module doesn't match other memory modules).
		08h	Spare. Indicates entity associated with the sensor represents a 'spare' unit of memory.
			The Event Data 3 field ~~for this command~~ can be used to provide an event extension code, with the following definition: 7:0 DIMM/SIMM/RIMM identification, relative to the entity that the sensor is associated with (if SDR provided for this sensor)

E286 Typos - Section 35.1, Sensor Type Code and Section 35.2, Event/Reading Type Code

The second sentence of the first paragraph of section 35.1 lists an incorrect value for the Sensor Type Code for processor. The value should be "07h" instead of "01h". This is corrected as follows:

> Each sensor has a *Sensor Type Code*. These codes are defined in *Table 36-3, Sensor Type Codes*. Sensor Type Codes are used both in SDRs and Event Messages. An example of a Sensor Type Code is code ~~01h~~07h, which

indicates a *Processor* sensor.

The same typo is repeated in section 35.2. under the item “Sensor Specific”.

Sensor Specific An Event/Reading Type Code of 6Fh indicates that the enumeration is defined explicitly for the particular Sensor Type. In an event message, an Event Dir bit of ‘0’ indicates the state has become asserted, while an Event Dir bit of ‘1’ indicates the state has become deasserted. For example, if the Sensor Type Code were ~~01h~~07h, “Processor”, then an Event/Reading Type Code of 6Fh would indicate that **…**

E287 Addendum - Support for Processor Throttling indications

A new sensor-specific offset for the Processor sensor type to indicate when a processor has entered or exited a throttled state that was automatically triggered by hardware.

Table 36-3, Sensor Type Codes

…

Processor	07h	00h	IERR
		01h	Thermal Trip
		02h	FRB1/BIST failure
		03h	FRB2/Hang in POST failure (used hang is believed to be due or related to a processor failure. Use System Firmware Progress sensor for other BIOS hangs.)
		04h	FRB3/Processor Startup/Initialization failure (CPU didn’t start)
		05h	Configuration Error
		06h	SM BIOS ‘Uncorrectable CPU-complex Error’
		07h	Processor Presence detected
		08h	Processor disabled
		09h	Terminator Presence Detected
		0Ah	Processor Automatically Throttled (processor throttling triggered by a hardware-based mechanism operating independent from system software, such as automatic thermal throttling or throttling to limit power consumption.)

E292 Addendum - Table 37-12, Entity ID Codes

There was no current Entity ID value to associate to the management controller firmware (as distinct from the OS-present software). Entity ID value of 46 (2Eh) has been added as the Entity ID for Management Controller Firmware.

Table 37-13, Entity ID Codes

Code		Entity
…		
46	2Eh	Management Controller Firmware (Represents firmware or software running on a management controller)
…		

E295 Addendum - Table 36-3, Sensor Type Code

A new sensor type is added to enable identifying and logging various types of changes that can occur to an Entity.

Table 36-3, Sensor Type Codes

...

Version Change	2Bh	00h	Hardware change detected with associated Entity. Informational. This offset does not imply whether the hardware change was successful or not. Only that a change occurred.
		01h	Firmware or software change detected with associated Entity. Informational. Success or failure not implied.
		02h	Hardware incompatibility detected with associated Entity.
		03h	Firmware or software incompatibility detected with associated Entity.
		04h	Entity is of an invalid or unsupported hardware version.
		05h	Entity contains an invalid or unsupported firmware or software version.
		06h	Hardware Change detected with associated Entity was successful. (deassertion event means 'unsuccessful').
		07h	Software or F/W Change detected with associated Entity was successful. (deassertion event means 'unsuccessful') *Event data 2 can be used for additional event information on the type of version change, with the following definition:* Event Data 2 7:0 Version change type 00h unspecified 01h management controller device ID (change in one or more fields from 'Get Device ID') 02h management controller firmware revision 03h management controller device revision 04h management controller manufacturer ID 05h management controller IPMI version 06h management controller auxiliary firmware ID 07h management controller firmware boot block 08h other management controller firmware 09h system firmware (EFI / BIOS) change 0Ah SMBIOS change 0Bh operating system change 0Ch operating system loader change 0Dh service or diagnostic partition change 0Eh management software agent change 0Fh management software application change 10h management software middleware change 11h programmable hardware change (e.g. FPGA) 12h board module change (change of a module plugged into associated entity) 13h board component change (addition or removal of a replaceable component not tracked as a FRU) 14h board replaced with equivalent version 15h board replaced with newer version 16h board replaced with older version 17h board hardware configuration change (e.g. strap, jumper, cable change, etc.)

...

E297 Addendum - Table 36-3, Sensor Type Codes, offsets for Power Supply Sensor

The following additional offset values have been added to the power supply sensor in Table 36-3, Sensor Type Codes to support the sensor type's use for DC-to-DC converters and processor voltage regulator modules (VRMs).

Table 36-3, Sensor Type Codes

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
			...
Power Supply (also used for power converters [e.g. DC-to-DC converters] and VRMs [voltage regulator modules]).	08h	00h	Presence detected
		01h	Power Supply Failure detected
		02h	Predictive Failure
		03h	Power Supply AC lost
		04h	AC lost or out-of-range
		05h	AC out-of-range, but present
		06h	Configuration error. The Event Data 3 field provides a more detailed definition of the error: 7:4 = Reserved for future definition, set to 0000b 3:0 = *Error Type*, one of 0h = Vendor mismatch, for ~~OEM-specified VRM~~power supplies that include this status. (Typically, the system OEM defines the vendor compatibility criteria that drives this status). 1h = Revision mismatch, for ~~OEM-specified VRM~~power supplies that include this status. (Typically, the system OEM defines the vendor revision compatibility that drives this status). 2h = Processor missing. For processor power supplies (typically DC-to-DC converters or VRMs), there's usually a one-to-one relationship between ~~a VRM~~the supply and ~~a~~the CPU. This offset can indicate the situation where the power supply is present but the processor is not. This offset can be used for reporting that as an unexpected or unsupported condition. Others = Reserved for future definition
			...

E298 Addendum and Clarification - Table 13-2, Serial Port Sharing Access Characteristics

The description of the mux setting while the Ring Time timeout was running was overly restrictive. The following wording changes 'loosen' up the requirement and retain backward compatibility with prior implementations. I.e. an implementation that met with the earlier stated requirements would remain conformant under these new descriptions. The same changes apply to the description for 'pre-boot' in Table 13-2.

Table 13-2, Serial Port Sharing Access Characteristics

…

enabled	disabled	Same behavior for both Modem and Direct Connect Mode • ~~Mux always set to system (except during alerting.~~ BMC takes connection during alerting. Otherwise, when power is on mux is set to system, when power is off mux setting is unspecified. • Escape sequence / pattern triggered switching is disabled. • Set Serial/Modem Mux command available. • Alerting available. • BMC Power-on Default = if system power is on mux set to system. If system power is off, mux setting is unspecified. BMC must take connection during alerting. BIOS Action at POST start: none required. BIOS Action at POST end: none required.

…

enabled	pre-boot	BMC pays attention to Modem Ring Time parameter until mux is *forced* to system using *Set Serial/Modem Mux* command. Afterwards, BMC will not automatically take over mux for IPMI messaging (will not answer the phone) until next power down or system reset (unless commanded). • Escape sequence / pattern triggered switching is available. • *Set Serial/Modem Mux* command available. • Alerting available. BMC will terminate call and automatically take the mux in order to send an alert, unless an IPMI Messaging Session is already in progress on the channel - in which case alert will be "deferred" until channel becomes available for dial-out. • BMC Power-on Default = ~~Mux set to system~~ ~~i~~If system power is off, or if Modem Ring Time >00h and <3Fh the power-on default mux setting is unspecified until RI or DCD is detected (see below), otherwise set to BMC. For Modem Mode, the BMC automatically takes over the connection upon power down, after system resets, and on detecting Ring based on Modem Ring Time parameter, except if a session is active - in which case the BMC will keep the connection (until the mux is *forced* to system using the *Set Serial/Modem Mux* command). If Modem Ring Time parameter is >00h and <3Fh, - If system power is on and the Ring Time countdown is running, the mux will be set to system to allow the system to answer the modem call. ~~Mux will be set to system.~~ BMC will take over mux if Ring Time expires while Ring is being detected via the RI signal. If system power is on, t~~T~~he mux will be returned to system when loss of connection (loss of DCD) is detected, or if the BMC takes the mux but is unable to establish a connection. If Ring Time = 00h, BMC will take mux during power down and after system resets as necessary to be able to answer the call via the modem. BMC will also take the mux and ~~answer phone~~ connect with the modem ~~immediately~~ when a Ring is detected via the RI signal. Mux will be claimed by the BMC whenever loss of DCD connection is detected. To the BMC, this is essentially the same 'phone answer' and power down/reset behavior as in 'Always Available' mode. For Direct Connect Mode the BMC automatically takes the connection upon power down, after system resets, and whenever loss of DCD is detected (if DCD-based switching is enabled) except if a session is active - in which case the BMC will keep the connection (until the mux is *forced* to system using the *Set Serial/Modem Mux* command). BIOS Action at POST start: Request mux to system if BIOS console redirection enabled. BIOS Action at POST end: Force to system. Keep baseboard serial controller enabled.

…

12/4/02 Addenda, Errata, and Clarifications

E300 Errata - Table 24-9, Alert Immediate Command - Errata Missing from IPMI v1.5 rev 1.1 Release

IPMI v1.5 rev 1.0 Errata 5 listed an E122 that should have been incorporated into the IPMI v1.5 rev. 1.1 release, but was missed. The command did not make provision to allow software to determine the status of the alert in progress. This has been corrected by defining a second response data byte and bits 7:6 in the Destination Selector field of the request data, as follows:

Table 24-9 Alert Immediate Command

	Byte	data field
Request Data	1	Channel number. (This value is required to select which configuration parameters are to be used to send the Alert.) [7:4] - reserved [3:0] - Channel number. Note: BMC stores the 'Alert immediate status' for each channel that can send alert.
	2	Destination Selector/ Operation [7:6] - Operation 00b = Initiate alert 01b = Get Alert Immediate status 10b = Clear Alert Immediate status (sets status to 00h) 11b = reserved [5:4] - Reserved [3:0] - destination selector. Selects which alert destination should go to. 0h = use volatile destination info. 1h-Fh = non-volatile destination. Note: If Operation is 'Get Alert Immediate status' or 'Clear Alert Immediate Status' bits [3:0] are reserved.
	3	Alert String Selector Selects which Alert String, if any, to use with the alert. [7] - 0b = don't send an Alert String 1b = send Alert String identified by following string selector. [6:0] - string selector. 000_0000b = use volatile Alert String. 01h-7Fh = non-volatile string selector.
Response Data	1	Completion Code. Generic codes, plus following command-specific completion codes: 81h = Alert Immediate rejected due to alert already in progress. 82h = Alert Immediate rejected due to IPMI messaging session active on this channel.
		Following byte is only returned when Operation in request is set to "Get Alert Immediate status"

2	Alert Immediate Status SMS can poll this status to determine present state of the immediate alert. 00h = No status. Note: A BMC implementation is allowed (but not required) to abort the Alert Immediate command due to a channel parameter configuration, power, or reset state changes that occur while the Alert Immediate command is being processed. In which case the BMC will return the 'no status' state. 01h = Alert was Normal End. This will also be returned if one or more attempts failed, but the last attempt was successful. 02h = "Call Retry" (Dial connection) retries failed. 03h = Alert failed due to timeouts waiting for acknowledge on all retries. FFh = Alert by this command is in progress. Status pending.

9/12/03 Addenda, Errata, and Clarifications

E276 Addendum and Clarification - Indicate when parameter changes take effect

The specification did not call out when changes to configuration parameters would take effect. The following section has been added to the IPMI Overview to address this, and to indicate that standardized configuration interfaces themselves are also an IPMI feature.

1.6.29 Configuration Interfaces

IPMI provides standardized interfaces and commands for configuring the platform management subsystem. This enables cross-platform software to Sensor Data Records are an example of the interface for configuring sensor population and behavior on a system. There are also commands for configuring capabilities such as LAN and serial/modem remote protocols, user passwords and privilege levels, Platform Event Filtering, alert destinations, and others.

Unless otherwise specified, changes to parameters are required to take effect for the next use. For example, parameters that affect user access or session operation must take effect for the next time a remote console attempts to connect to the system. In some implementations, changes to configuration parameters may take effect immediately. Thus, a remote application should be careful when setting parameters that could cause the application to become disconnected from the BMC.

For the purpose of conformance checking, up to 5 seconds will be allowed between the time a parameter is changed to when it must have taken effect.

It is recognized that there are race conditions where a session may already be in the process of being established before the change can be propagated. It is recommended that a BMC implementation takes steps to ensure that parameters are used consistently. This specification does not define a specific mechanism, but here are some possible approaches. An implementation could terminate a session in progress if the user's parameters change while the session is being established. Alternatively, an implementation could 'snap shot' the user's configuration at the time the session is being established and only allow a session to be established if the given user's configuration has been unmodified in the last 5 seconds.

E285 Errata - Table 18-17, Activate Session Command

The intent is that a packet can always be authenticated and accepted, even if per-message authentication is disabled. This is required in order to always allow an authenticated message to be delivered to system management software via the receive message queue (using the Send Message command). The Activate Session command included text that incorrectly indicated that the BMC would *only* accept packets with Authentication Type = 'none' when per-message authentication was disabled. This has been corrected as follows:

Table 18-17, Activate Session Command

	byte	data field
Session Request Data		authentication type = from corresponding *Get Session Challenge* command.
	...	
IPMI Response Data	1	Completion Code 00h = success 81h = No session slot available (BMC cannot accept any more sessions) 82h = No slot available for given user. (Limit of user sessions allowed under that name has been reached) 83h = No slot available to support user due to maximum privilege capability. (An implementation may only be able to support a certain number of sessions based on what authentication resources are required. For example, if User Level Authentication is disabled, an implementation may be able to allow a larger number of users that are limited to User Level privilege, than users that require higher privilege.) 84h = session sequence number out-of-range 85h = invalid session ID in request 86h = requested maximum privilege level exceeds user and/or channel privilege limit
	2	Authentication Type for remainder of session The primary use of this parameter is to report whether per-message authentication will be used for IPMI message packets that follow the *Activate Session* packet. Per-message authentication is a channel configuration option that is set using the *Get User Name* command. If per-message authentication is disabled, the Authentication Type will be returned as 'none', and all subsequent packets for the session ~~are required to~~ can either use 'none' as the authentication type or use the Authentication Type that was used in the request. Otherwise this value will be set to the Authentication Type that was used in the request. Note that *Activate Session* requests and responses are always required to be authenticated per what is returned by the *Get Session Challenge* command for the user. [7:4] - reserved [3:0] - Authentication Type 0h = none. No hashing or authentication done on session packets. Authentication Code field is not present. 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary all other = reserved
	...	

E293 Addendum - ICMB group power control support

The following parameters and changes have been added to support ICMB v1.3 Group Control capabilities.

The following parameters have been added to Table 24-6, PEF Configuration Parameters:

Table 24-6, PEF Configuration Parameters

Parameter	#	Parameter Data
		...
Number of Group Control Table entries (optional. Present if BMC supports automatic ICMB Group Power Control. See ICMB specification for details.)	14	data 1 - Number of group control table entries. 1-based (4 min, 8 max)
Group Control Table (optional, non-volatile. Present if BMC supports automatic ICMB Group Power Control. See ICMB specification for details.)	15	data 1 - Set Selector = group control table entry selector. [7:4] - reserved. [3:0] - group control table entry selector. 1-based. 0h = reserved. data 2 - [7:6] - reserved [5] - Request/Force 0b = request control operation. A requested operation will only complete once the same operation has been requested for all control groups and all enabled control members for the given chassis. 1b = force control operation. A forced operation will occur regardless of whether the same operation has been requested for all control groups and all enabled control membership for the given chassis. [4] - Immediate/Delayed. Selects whether the BMC requests an immediate or delayed control operation. Note: whether this operation is initiated at the time the command is received is dependent on the request/force bit, see above. 0b = immediate control. BMC sends command that requests an immediate control operation. 1b = delayed control. BMC sends control command to request a delayed control operation. This is conditioned by the request/force bit. [3:0] - Channel Number (channel number for ICMB that group control operation is to be delivered over) data 3: Group ID 0 (1-based) 00h = unspecified FFh = all groups data 4: Member ID 0 (0-based) [7:5] - reserved [4] - 0b = enable member ID check. 1b = disable member ID check[1]. [3:0] - member ID. ID of this chassis within specified group. (value is ignored if Group ID 0 = FFh) data 5: Group ID 1 (1-based) 00h = unspecified FFh = all groups data 6: Member ID 1 (0-based) [7:5] - reserved [4] - 0b = enable member ID check. 1b = disable member ID check[1].

		[3:0] - member ID. ID of this chassis within specified group. (value is ignored if Group ID 1 = FFh) data 7: Group ID 2 (1-based) 00h = unspecified FFh = all groups data 8: Member ID 2 (0-based) [7:5] - reserved [4] - 0b = enable member ID check. 1b = disable member ID check[1]. [3:0] - member ID. ID of this chassis within specified group. (value is ignored if Group ID 2 = FFh) data 9: Group ID 3 (1-based) 00h = unspecified FFh = all groups data 10: Member ID 3 (0-based) [7:5] - reserved [4] - 0b = enable member ID check. 1b = disable member ID check[1]. [3:0] - member ID. ID of this chassis within specified group. (value is ignored if Group ID 3 = FFh) data 11: - Retries and Operation [7] - reserved [6:4] - number of times to retry sending the command to perform the group operation [For ICMB, the BMC broadcasts a *Group Chassis Control* command] (1-based) [3:0] - operation 0h = power down. Force system into soft off (S4/S45) state. This is for 'emergency' management power down actions. The command does not initiate a clean shut-down of the operating system prior to powering down the system. 1h = power up. 2h = power cycle (optional). This command provides a power off interval of at least 1 second. 3h = hard reset. Some systems may accept this option even if the system is in a state (e.g. powered down) where resets are unavailable. 4h = pulse Diagnostic Interrupt. (optional) Pulse a version of a diagnostic interrupt that goes directly to the processor(s). This is typically used to cause the operating system to do a diagnostic dump (OS dependent). The interrupt is commonly an NMI on IA-32 systems and an INIT on Intel® Itanium™ processor based systems. 5h = Initiate a soft-shutdown of OS via ACPI by emulating a fatal overtemperature. (optional)

...

The following has been added to Table 15-2, Event Filter Table Entry:

Table 15-2, Event Filter Table Entry

Byte	Field	Description
2	Event Filter Action	All actions are optional for an implementation, with the exception of Alert which is mandatory if alerting is supported for one or more channels. The BMC will return 0b for unsupported actions. Software can test for which actions are supported by writing 1's to the specified fields and reading back the result. (Note that reserved bits must be written with 0's) [7~~:6~~] - reserved [6] - 1b = group control operation 0b = no group control operation [5] - 1b = Diagnostic Interrupt (NMI) 0b = no Diagnostic Interrupt [4] - 1b = OEM action 0b = no OEM [3] - 1b = power cycle 0b = no power cycle [2] - 1b = reset 0b = no reset [1] - 1b = power off 0b = no power off [0] - 1b = Alert 0b = no Alert
3	Alert Policy Number	Used to select an alerting policy set from the Alert Policy Table. The Alert Policy Table holds different policies that configure the order in which different alert destinations and alerting media are tried. [7~~:4~~] - reserved [7:4] - group control selector (1-based). 0h=reserved. Selects entry from group control table. [3:0] - policy number. Value is 'don't care' if Alert is not selected in the Event Filter Action.

E294 Addendum - Table 36-3, Sensor Type Codes

This addendum is an extension to Memory Sensor (10h) offset 00h "Correctable Memory Error Logging Disabled" to provide the option of an Event Data 2 field holding a numeric ID for the memory device/module for which correctable error logging has been disabled.

Table 36-3, Sensor Type Codes

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
			...
Event Logging Disabled	10h	00h	Correctable Memory Error Logging Disabled Event Data 2 [7:0] - Memory module/device (e.g. DIMM/SIMM/RIMM) identification, relative to the entity that the sensor is associated with (if SDR provided for this sensor).
		01h	Event 'Type' Logging Disabled. Event Logging is disabled for following event/reading type and offset has been disabled. Event Data 2 Event/Reading Type Code Event Data 3 [7:6] - reserved. Write as 00b. [5] - 1b = logging has been disabled for all events of given type [4] - 1b = assertion event, 0b = deassertion event [3:0] - Event Offset
		02h	Log Area Reset/Cleared
		03h	All Event Logging Disabled
			...

E296 Addendum - PCI Class Codes for IPMI System Interfaces

PCI Class Codes have been reserved for IPMI System Interfaces implemented on PCI. This has been documented in the changes for E318.

Also added in E318 that PCI implementations of the KCS interface that are not byte-aligned must return a fixed 00h in the unused byte positions. This will help enable a driver to test for alignment instead of having to rely solely on the SMBIOS record.

E301 Addendum and Clarification - Table 13-2, Serial Port Sharing Access Characteristics

This erratum adds to E298. The description of the mux setting while the Ring Time timeout was running was overly restrictive. E298 did not cover the case where Serial Port Sharing = disabled, and Access Mode = disabled. The following wording changes 'loosen' up the requirement and retain backward compatibility with prior implementations. I.e. an implementation that met with the earlier stated requirements would remain conformant under these new descriptions.

Table 13-2, Serial Port Sharing Access Characteristics

Serial Port Sharing	Access Mode	Characteristics
disabled	disabled	Same behavior for both Modem and Direct Connect Mode • If system power is On, Mux always set to system. When power is off Mux setting is unspecified. • Set Serial/Modem Mux command is rejected. (See response data for the Set Serial/Modem Mux command). • Escape sequence / pattern triggered switching is not available. • Alerting Unavailable. • BMC Power-on Default = mux set to system when system power is On. When power is off, mux setting is unspecified. BIOS Action at POST start: none required BIOS Action at POST end: none required

…

E302 Typo - Table 37-3, Event-Only Sensor Record - SDR Type 03h

In the sensor data record definitions, byte 7 Sensor Owner LUN, bits 1:0 are defined as:

"Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive IPMB messages to access the sensor. 00b if system software is Sensor Owner."

It should say "IPMI messages" instead of IPMB messages. This is corrected as follows:

Table 37-3, Event-Only Sensor Record - SDR Type 03h

…

7	Sensor Owner LUN	1	[7:4] - Channel Number The Channel Number can be used to specify access to sensors that are located on management controllers that are connected to the BMC via channels other than the primary IPMB. (Note: In IPMI v1.5 the ordering of bits 7:2 of this byte have changed to support the 4-bit channel number) [3:2] - FRU Inventory Device Owner LUN. LUN for Write/Read FRU commands to access FRU information. 00b otherwise. [1:0] - Sensor Owner LUN. LUN in the Sensor Owner that is used to send/receive ~~IPMB~~ IPMI messages to access the sensor. 00b if system software is Sensor Owner.

…

E303 Addendum - Table 37-12, Entity ID Codes

An Entity ID Code for IPMI Channels was added to enable sensors to be associated with them for the purpose of reporting physical status of the channels.

Table 37-13, Entity ID Codes

Code		Entity

…

47	2Fh	IPMI Channel - This Entity ID enables associating sensors with the IPMI communication channels - for example a Redundancy sensor could be used to report redundancy status for a channel that is composed of multiple physical links. By convention, the Entity Instance corresponds to the channel number.

…

E304 Clarification - Table 37-4, Entity Association Record - SDR Type 08h, and Table 37-5, Device-relative Entity Association Record - SDR Type 09h

The clarification changes the '00h=unspecified' notes for the Entity Instance fields as follows. The original text could be misinterpreted as implying that 00h is not a legal value for an Entity Instance.

Table 37-4, Entity Association Record - SDR Type 08h

byte	Field Name	size	Description
	RECORD HEADER		
…			
	RECORD BODY BYTES		
11	Contained Entity 2 / Range 1 entity	1	00h = unspecified If list: Entity ID for contained entity 2 If range: Entity ID of entity for contained entity range 1 (must match byte 9)
12	Contained Entity 2 Instance / Range 1 last entity Instance	1	~~00h = unspecified~~Set to 00h if Entity 2 is unspecified If list: Instance ID for contained entity 2 If range: Instance ID for last entity in contained entity range 1
13	Contained Entity 3 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 3 If range: Entity ID of entity for contained entity range 2
14	Contained Entity 3 Instance / Range 2 first entity Instance	1	~~00h = unspecified~~Set to 00h if Entity 3 is unspecified If list: Instance ID for contained entity 3 If range: Instance ID for first entity in contained entity range 2
15	Contained Entity 4 / Range 2 entity	1	00h = unspecified If list: Entity ID for contained entity 4 If range: Entity ID for entity for contained entity range 2 (must match byte 13)
16	Contained Entity 4 Instance / Range 2 last entity Instance	1	~~00h = unspecified~~Set to 00h if Entity 4 is unspecified If list: Instance ID for contained entity 4 If range: Instance ID for last entity in contained entity range 2

Table 37-5, Device-relative Entity Association Record - SDR Type 09h

byte	Field Name	size	Description
	RECORD HEADER		
…			
18	Contained Entity 2 Instance / Range 1 last Entity Instance	1	Set to 00h ~~=~~ if Entity 2 is unspecified If list: Entity Instance for contained entity 2 If range: Entity Instance for last entity in contained entity range 1
…			
22	Contained Entity 3 Instance / Range 2 first entity Instance	1	Set to 00h ~~=~~ if Entity 3 is unspecified If list: Entity Instance for contained entity 3 If range: Entity Instance for first entity in contained entity range 2
…			
26	Contained Entity 4 Instance / Range 2 last entity Instance	1	Set to 00h if Entity 4 is~~=~~ unspecified If list: Entity Instance for contained entity 4 If range: Entity Instance for last entity in contained entity range 2
…			

E305 Clarification - "Initial update in progress" → "Reading/state not available"

The bit name has been globally changed to "Reading/state not available". Text has been added to indicate that the bit can be used anytime that the BMC needs to indicate to software that the sensor is present,

but that the reading and/or state information for the sensor is not available. Software should ignore the reading and/or state information from a sensor when this bit is set.

The following text has been added as the last paragraph of section 14.4:

14.4 Event Status, Event Conditions, and Present State

A sensor tracks present state and *Event Conditions*. An Event Condition is that set of comparisons applied to the present state and previous state that produces a given *Event Status*.

…

A *Reading/State Unavailable (formerly "Initial update in progress")* bit is provided with the *Get Sensor Reading* and *Get Sensor Event Status* commands to indicate to software that it must ignore the reading and/or state information because the BMC cannot obtain a valid reading and/or state information. This can occur in situations where the sensor is monitoring an entity that may or may not be present, such as with hot-swap devices. For example, if a sensor monitors the temperature of a hot-swap power supply, the Reading/State Unavailable bit can be used to indicate that no valid temperature reading is available because the power supply is not installed. The bit can also indicate when a reading or state is unavailable because a sensor is re-arming (see *Section 14.6, Re-arming*, below.

The following change was made to section 14.6:

14.6 Re-arming

Re-arm refers to resetting internal device state that tracks that an event has occurred on the sensor. After a sensor is re-armed the device will re-check the event condition and re-generate the event if the event condition exists.

If the event condition already exists at the time that the re-arm is initiated, then it is possible that the event will be regenerated immediately following the conclusion of the re-arm. A~~n *initial update in progress*~~*Reading/State Unavailable (formerly "Initial Update in progress")* bit is provided with the *Get Sensor Reading* and *Get Sensor Event Status* commands to help software avoid getting incorrect event status due to a re-arm.

The following changes were made to the note in section 23.1 and last paragraph of section 29.12:

23.1 Set Event Receiver Command

…

⇒ A ~~*initial update in progress*~~*reading/state unavailable (formerly "initial update in progress")* bit is provided with the *Get Sensor Reading* and *Get Sensor Event Status* commands to help software avoid getting incorrect event status due to a re-arm. For example, suppose a controller only scans for an event condition once every four seconds. Software that accessed the event status using the *Get Sensor Reading* command could see the wrong status for up to four seconds before the event status would be correctly updated. A controller that has slow updates must implement the ~~initial update in progress bit~~*reading/state unavailable bit*, and should not generate event messages until the update has completed. Software should ignore the Event Status bits while the ~~*initial update in progress*~~*reading/state unavailable* bit is set.

29.12 Re-arm Sensor Events Command

…

A ~~initial update in progress~~reading/state unavailable (formerly "initial update in progress") bit is provided with the Get Sensor Reading and Get Sensor Event Status commands to help software avoid getting incorrect event status due to a re-arm. For example, suppose a controller only scans for an event condition once every four seconds. Software that accessed the event status using the Get Sensor Reading command could see the wrong status for up to four seconds before the event status would be correctly updated. A controller that has slow updates must implement the initial update in progress bit, and should not generate event messages until the update has completed. Software should ignore the Event Status bits while the ~~initial update in progress~~*reading/state unavailable* bit is set.

Lastly, the following changes were made to Tables 29-14 and 29-15:

Table 29-14, Get Sensor Event Status Command

Request Data	1	Sensor number (FFh = reserved)
Response Data	1	Completion Code
	2	[7] - 0b = All Event Messages disabled from this sensor [6] - 0b = Sensor scanning disabled [5] - 1b = ~~initial update in progress~~reading/state unavailable (formerly "initial update in progress"). This bit is set to indicate that a 're-arm' or 'Set Event Receiver' command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a 'Get Sensor Reading' or 'Get Sensor Event Status' command for the sensor before the update has completed. This is most likely to be the case for sensors, such as fan RPM sensors, that may require seconds to accumulate the first reading after a re-arm. The bit is also used to indicate when a reading/state is unavailable because the management controller cannot obtain a valid reading or state for the monitored entity, typically because the entity is not present. See *Section 14.4, Event Status, Event Conditions, and Present State* and *Section 14.6, Re-arming* for more information. [4:0] - reserved. Ignore on read.

…

Table 29-15, Get Sensor Reading Command

Request Data	1	sensor number (FFh = reserved)
Response Data	1	Completion Code.
	2	Sensor reading Byte 1: byte of reading. Ignore on read if sensor does not return an numeric (analog) reading.
	3	[7] - 0b = All Event Messages disabled from this sensor [6] - 0b = sensor scanning disabled [5] - 1b = ~~initial update in progress~~reading/state unavailable (formerly "initial update in progress"). This bit is set to indicate that a 're-arm' or 'Set Event Receiver' command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a 'Get Sensor Reading' or 'Get Sensor Event Status' command for the sensor before the update has completed. This is most likely to be the case for sensors, such as fan RPM sensors, that may require seconds to accumulate the first reading after a re-arm. The bit is also used to indicate when a reading/state is unavailable because the management controller cannot obtain a valid reading or state for the monitored entity, typically because the entity is not present. See *Section 14.4, Event Status, Event Conditions, and Present State* and *Section 14.6, Re-arming* for more information. [4:0] - reserved. Ignore on read.

...

E307 Errata - Table 36-3, Sensor Type Codes

For the Power Supply sensor type, the term "AC lost" should mean "Power Supply input lost" - since the Power Supply sensor type applies to DC-to-DC converters as well as AC-to-DC power supplies. T

Table 36-3, Sensor Type Codes

...

Power Supply (also used for power converters [e.g. DC-to-DC converters] and VRMs [voltage regulator modules]).	08h	00h	Presence detected
		01h	Power Supply Failure detected
		02h	Predictive Failure
		03h	Power Supply input lost (AC / DC)[2]
		04h	Power Supply input ~~AC~~ lost or out-of-range
		05h	Power Supply input ~~AC~~ out-of-range, but present
		...	...

...

2. "Power supply input" refers to AC or DC source inputs to the power supply or power converter.

E308 Addendum - Option to allow power cycle interval to be set

A new optional command has been added to provide a common way to enable the power cycle interval for a platform to be set.

22.7a Set Power Cycle Interval

This command provides a mechanism for configuring the power cycle interval for the system. This interval determines the time that system power will be powered down during a power cycle operation initiated by the *Chassis Control* command or the watchdog time. The setting is non-volatile.

Table 22-8a, Set Power Cycle Interval Command

	byte	data field
Request Data	1	[7:0] - Power Cycle Interval in seconds. 1-based. Non-volatile. 00h = no delay.
Response Data	1	Completion code.

The following supporting additions have been made to the command tables as follows:

Table 22-1, Chassis Commands

Command	Section Defined	O/M
…		
Set Power Cycle Interval	22.7a	O
…		

Table G-1, Command Number Assignments and Privilege Levels

	section	NetFn	CMD	C	U	O	A
…							
Set Power Cycle Interval	22.7a	Chassis	0Bh				X
…							

E310 Clarification and Addendum - Section 29.2, Get Device SDR Info Command, and Section 29.3, Get Device SDR Command

A clarification was added that the *Get Device SDR* command can be used to return any type of SDR, not just Types 01h and 02h. The *Get Device SDR Info* command returns the number of sensors rather than the number of Sensor Data Records. An optional request parameter has been added to allow the command to return the total number of sensor data records as well.

29.2 Get Device SDR Info Command

This command returns general information about the collection of sensors in a Dynamic Sensor Device.

Note: If the command is issued with no parameter for the request, the Device Sensor information is LUN based. That is, it is returned individually for each LUN. E.g.. a device could implement four sensors under one LUN, and twelve under another. The SDR Info does *not* return the aggregate of the sensor information. Rather, separate 'Get Device SDR Info' commands need to be issued to each LUN. The 'Device LUNs' field is provided in the response to support this.

Software should assume an allocation unit size of 16-bytes ~~if this command is not implemented~~for device SDRs.

Table 29-2, Get Device SDR Info Command

Request Data	-(1)	Operation (optional) -[7:1] - reserved [0] - 1b = Get SDR count. This returns the total number of SDRs in the device. 0b = Get Sensor count. This returns the number of sensors implemented on LUN this command was addressed to.
Response Data	1	Completion Code
	2	For Operation = "Get Sensor Count" (or if byte 1 not present in request): Number of sensors in device for LUN this command was addressed to. For Operation = "Get SDR Count": Total Number of SDRs in the device.
	3	Flags: Dynamic population [7] - 0b = static sensor population. The number of sensors handled by this device is fixed, and a query shall return records for all sensors. 1b = dynamic sensor population. This device may have its sensor population vary during 'run time' (defined as any time other that when an install operation is in progress). Reserved [6:4] - reserved Device LUNs [3] - 1b = LUN 3 has sensors [2] - 1b = LUN 2 has sensors [1] - 1b = LUN 1 has sensors [0] - 1b = LUN 0 has sensors
	4:7	Sensor Population Change Indicator. LS byte first. Four byte timestamp, or counter. Updated or incremented each time the sensor population changes. This field is not provided if the flags indicate a static sensor population.

29.3 Get Device SDR Command

The 'Get Device SDR' command allows ~~the population of sensors for the given LUN in the device to be listed and the associated~~ SDR information for ~~those sensors~~ sensors for a Sensor Device (typically implemented in a satellite management controller) to be returned. The *Get Device SDR Command* can return any type of SDR, not just Types 01h and 02h. This is an optional command for Static Sensor Devices, and mandatory for Dynamic Sensor Devices. The format and action of this command is similar to that for the 'Get SDR' command for SDR Repository Devices. **…**

E312 Addendum - Table 22-9, Get System Restart Cause Command

Added "RTC Wakeup" as a system restart cause, as follows:

Table 22-9, Get System Restart Cause Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	2	Restart Cause [7:4] - reserved [3:0] - 0h = unknown (system start/restart detected, but cause unknown) [required if this condition exists] 1h = Chassis Control command [required] 2h = reset via pushbutton [optional] 3h = power-up via power pushbutton [optional] 4h = Watchdog expiration (see watchdog flags) [required] 5h = OEM [optional] 6h = automatic power-up on AC being applied due to 'always restore' power restore policy (see *22.7, Set Power Restore Policy* Command) [optional] 7h = automatic power-up on AC being applied due to 'restore previous power state' power restore policy (see *22.7, Set Power Restore Policy* Command) [optional] 8h = reset via PEF [required if PEF reset supported] 9h = power-cycle via PEF [required if PEF power-cycle supported] Ah = soft reset (e.g. CTRL-ALT-DEL) [optional] Bh = power-up via RTC (system real time clock) wakeup [optional] all other = reserved
	3	Channel number. (Channel that command was received over)

E314 Addendum - Table 36-3, Sensor Type Codes

Added an offset "04h" to the Event Logging Disabled sensor to indicate that the event log is full.

Table 36-3, Sensor Type Codes

Sensor Type	Sensor Type Code	Sensor-specific Offset	Event
			...
Event Logging Disabled	10h	... 04h	... SEL Full
			...

E315 Clarification - Section 6.12.4, Bridged Request Example

Added text to clarify how responses are returned to a request that was bridged to another channel using the *Send Message* command when the source of the message is a channel other than the system interface (e.g. a request bridged from LAN to IPMI using *Send Message*).

6.12.4 Bridged Request Example

The example illustrates a *Send Message* command from LAN being used to deliver a request to IPMB.

Bridged requests to the IPMB can come from several different channels: LAN, serial/modem, and the ICMB. The BMC uses the sequence number that it places on the bridged request to identify which channel and to which address on that channel the response is to go back to. It is therefore important for the BMC to ensure that unique sequence numbers are used for pending requests from the different channels. It is also important that sequence numbers are unique for

successive requests to a given responder. One way to manage sequence numbers to the IPMB is to track sequence numbers on a per responder basis. This can be kept in a table of 'Pending Bridged Response' info.

In order to get the response back to the LAN, the IPMB response must return the same sequence number that was passed in the request. (This is a basic rule of IPMI Messaging, so there's nothing special about that requirement.) The management controller uses the sequence number to look up the channel type specific addressing, sequence number, and security information that it stored when the request was forwarded. For example, if the channel type is 'LAN' then the response message must be formatted up in an RMCP/UDP packet with the IP address of the requester, the sequence number passed in the original request, the appropriate security 'key' information, etc.

When a request message is bridged to another channel by encapsulating it in a *Send Message* command (from a source channel other than the system interface), the BMC immediately returns a response to the *Send Message* command itself. Meanwhile, the request is extracted from the *Send Message* command and forwarded to the specified target channel.

The *Send Message* command must be configured to direct the BMC to keep track of data in the request so when the response comes back from the target device it can be forwarded by the BMC back to the channel that delivered the original *Send Message* command to the BMC. When the response comes back from the target, the BMC uses the tracking information to format the response for the given channel. To the party that initiated the *Send Message* command, the response will appear as if the encapsulated request was directly executed by the BMC. I.e. it will look like an asynchronously generated response message.

For example, suppose a *Get Device ID* command has been encapsulated in a *Send Message* command directed to the IPMB from a LAN channel. The BMC will immediately send a response to the *Send Message* command back on LAN. The BMC will extract the encapsulated *Get Device ID* message content and format it as a *Get Device ID* request for IPMB. The target device on IPMB responds with a *Get Device ID* response message in IPMB format. The BMC takes the tracking information that was stored when the *Send Message* command was issued, and uses it to create a *Get Device ID* response in LAN format. The Responder's address information in that response will be that of the BMC, not of the device on IPMB that the request was targeted to.

The following figure and steps present an example high-level design for handling a bridged request. Note that the example shows information that is generated and stored, but it does not show any particular code module that would perform that operation. That is, the choice of which functions are centralized, which are in a 'LAN' module, and which are in an 'IPMB' module (or whether you even have such modularity) is left to the implementer.

E316 Addendum - Section 22.5, Chassis Identify Command

The *Chassis Identify* command has been extended with an optional parameter that, if supported, allows software to command the *Chassis Identify* indication to go on indefinitely.

22.5 Chassis Identify Command

This command causes the chassis to physically identify itself by a mechanism chosen by the system implementation; such as turning on blinking user-visible lights or emitting beeps via a speaker, LCD panel, etc. Unless the optional "Force Identify On" capability is supported and used, the *Chassis Identify* command automatically times out and deasserts the indication after a configurable time-out. Software must periodically resend the command to keep the identify condition asserted. This will restart the timeout.

Table 22-6, Chassis Identify Command

	byte	data field
Request Data	(1)[1]	[7:0] - Identify Interval in seconds. 1-based. Timing accuracy = -0/+20%. This field is optional. If this byte is not provided the default timeout shall be 15 seconds -0/+20%. Note that this byte can be overridden by optional byte 2. 00h = Turn off Identify
	(2)[2]	Force Identify On. This optional field enables software to command the Identify to be On indefinitely. The BMC implementation should return an error completion code if this byte is not supported. [7:1] - reserved [0] - 1b = Turn on Identify indefinitely. This overrides the values in byte 1. 0b = Identify state driven according to byte 1.
Response Data	1	Completion Code

1. This parameter byte is optionally present. If not provided, the *Chassis Identify* can be used to turn on the Identify indication for the default timeout interval, but cannot be used to turn the indication off.
2. This parameter byte is optionally present. If provided, it is highly recommended that the chassis provides a local manual mechanism that enables a user or service personnel to turn off Identify. If a local manual mechanism is not provided, AC removal (BMC reset) should remove the indication.

E317 Addendum - Section 22.2, Get Chassis Status Command, plus new command to support front panel lockout

The *Get Chassis Status* command is updated to include a new, optional 5th byte that can return front panel lockout status, as follows:

22.2 Get Chassis Status Command

The following command returns information regarding the high-level status of the system chassis and main power subsystem.

Table 22-3, Get Chassis Status Command

	byte	data field
Request Data	-	-
Response Data	1	Completion Code
	...	
	(5)	Front Panel Button capabilities and disable/enable status (Optional) ("Button" actually refers to the ability for the local user to be able to perform the specified functions via a pushbutton, switch, or other 'front panel' control built into the system chassis.) [7] - 1b = Standby (sleep) button disable allowed [6] - 1b = Diagnostic Interrupt button disable allowed [5] - 1b = Reset button disable allowed [4] - 1b = Power off button disable allowed (in the case there is a single combined power/standby (sleep) button, disabling power off also disables sleep requests via that button.) [3] - 1b = Standby (sleep) button disabled [2] - 1b = Diagnostic Interrupt button disabled [1] - 1b = Reset button disabled [0] - 1b = Power off button disabled (in the case there is a single combined power/standby (sleep) button, then this indicates that sleep requests via that button are also disabled.)

1. In some installations, the chassis' main power feed may be DC based. For example, -48V. In this case, the power restore policy for AC/mains refers to the loss and restoration of the DC main power feed.

A new *Set Front Panel Button Enables* command is defined to support 'front panel button lockout' capabilities, as follows:

22.2b Set Front Panel Button Enables Command

The following command is used to enable or disable the buttons on the front panel of the chassis. (Button actually refers to the ability for the local user to be able to perform the specified functions via a pushbutton, switch, or other 'front panel' control built into the system chassis.) These values will be returned in the Front Panel Button capabilities and disable/enable status (byte 5) of the *Get Chassis Status* command.

Table 22-6a, Set Front Panel Button Enables Command

	byte	data field
	1	Front Panel Button Enables [7:4] - reserved [3] - 1b = disable Standby (sleep) button for entering standby (sleep) (control can still be used to wake the system) [2] - 1b = disable Diagnostic Interrupt button [1] - 1b = disable Reset button [0] - 1b = disable Power off button for power off only (in the case there is a single combined power/standby (sleep) button, then this also disables sleep requests via that button)
Response Data	1	Completion Code

Table 22-1 and the command Table G-1 are updated to reflect the addition of the new command, as follows:

Table 22-1, Chassis Commands

Command	Section Defined	O/M
Get Chassis Capabilities	22.1	M
…		
Chassis Identify	22.5	O
Set Front Panel Enables	22.5a	O
Set Chassis Capabilities	22.6	O
…		

Table G-1, Command Number Assignments and Privilege Levels

	section	NetFn	CMD	C	U	O	A
…							
Chassis Device Commands							
Get Chassis Capabilities	22.1	Chassis	00h		X		
Get Chassis Status	22.2	Chassis	01h		X		
Chassis Control	22.3	Chassis	02h			X	
Chassis Reset	22.4	Chassis	03h			X	
Chassis Identify	22.5	Chassis	04h			X	
Set Front Panel Button Enables	22.5a	Chassis	0Ah				X
…							

E318 Addendum - Appendices and Sections on Describing IPMI System Interfaces on PCI and using ACPI

This addendum adds two new appendices to the IPMI v1.5 specification, Appendix C2 and C3. These present the PCI Class codes for an IPMI System Interface on PCI and describe how to present system interface resources via ACPI, respectively. In addition, Section 6.10a has been added to describe how system interfaces and BMCs with multiple system interfaces are discovered and supported.

The new Section 6.10a and appendices follows:

6.10a System Interface Discovery and Multiple Interfaces

A BMC device may make available multiple system interfaces, but only one management controller is allowed to be the 'active' BMC that provides BMC functionality for the system (in the case of a 'partitioned' system, there can only one active BMC per partition). Only the system interface(s) for the active BMC are allowed to respond to the *Get Device ID* command. If other BMC devices are present, but not being used, they must not respond to the *Get Device ID* command.

When system interfaces are available, the driver can select the type interface it wishes to use.

Drivers should not switch system interfaces during system operation or else unexpected results could occur. The *Get Device ID* command is required to execute correctly across multiple interfaces to a BMC, but other commands are not. Once the driver has chosen to use a given interface, all commands beyond *Get Device ID* should be delivered to that interface. If it is desired to change the choice of system interfaces, a warm or cold reset of the platform should be done to ensure that the system can re-initialize BMC operation.

It is recommended that run-time drivers support the IPMI System Interfaces in the following order:

- A driver should preferentially use the BMC on PCI, via the OS's native support, if available. (A "Plug and Play" OS will typically locate and load the appropriate driver for devices it finds on PCI.) *Appendix C2, Locating IPMI System Interfaces on PCI*, summarizes the PCI Class codes for IPMI System Interfaces.
- If the desired interface is not available on PCI, or the system is in a state where OS support for PCI is unavailable the next step should be to look for the system interface as a static resource described in ACPI using the control methods described in *Appendix C3, Locating IPMI System Interfaces with ACPI*.
- If the operating environment does not include a mechanism to support executing ACPI control methods, then look for the system interface at the location described by the SPMI (Service Processor Management Interface) Table(s) through the ACPI Description Table mechanisms. (The SPMI Table approach supports BMCs that offer more than one system interface. Therefore, there can be more than one instance of the SPMI Table.) The SPMI Table is described in *Appendix C3, Locating IPMI System Interfaces on PCI.*
- If the SPMI Table is not present, the driver should look for the SMBIOS Type 38 table (See *Appendix C1*) and use the interface described there. Unlike the SPMI Table, there is only one instance of the Type 38 record allowed, so the driver will not need to look for additional interfaces.
- Lastly, the driver should look for the IPMI System Interface at the fixed, default I/O addresses specified for the SMIC, KCS, and BT interfaces. Refer to the individual sections on those interfaces for the addresses.

Appendix C2 - Locating IPMI System Interfaces on PCI

The PCI SIG (http://www.pcisig.com) has defined class codes for IPMI System Interfaces in Appendix D of the *PCI Local Bus Specification,Revision 2.3, March 29, 2002*. PCI-based implementations of the IPMI System Interfaces should use the

appropriate PCI configuration space and the class code definition there to report the presence and type of system interface for driver loading purposes.

A BMC is allowed to support more than one type of system interface simultaneously. It is possible Only an active BMC should respond to the *Get Device ID* command.

The first base address register of the PCI function holding the IPMI System Interface. The IPMI System Interfaces can be I/O or memory mapped, as indicated by read-only bits in the base address register.

Unless otherwise specified, IPMI System Interfaces on PCI must be byte aligned and located at offset 0 with respect to the base address register. PCI implementations of the KCS interface that are not byte-aligned must return a fixed 00h in the unused byte positions. This enables a driver to test for alignment. Non- byte-aligned KCS interfaces must also have their eight-bit registers aligned on even 32-bit or 16-byte boundaries starting at offset 0 with respect to the base address register.

Table C2-1, PCI Class Codes for IPMI

Class Code	Sub Class	Interface	Description
0Ch			Serial Bus Controllers (Historically, the IPMI System Interfaces were defined under this class because of the use of BMCs as interfaces to serial busses such as private management busses and the IPMB)
	07h		IPMI System Interfaces
		00h	IPMI SMIC Interface
		01h	IPMI Keyboard Controller Style (KCS) Interface
		02h	IPMI Block Transfer (BT) Interface

Appendix C3 - Locating IPMI System Interfaces with ACPI

Revision 1.2 of the IPMI v1.5 specification introduces the option of describing the presence of the IPMI System Interface as a static (non- “Plug and Play”) resource using ACPI. The IPMI System Interface can also be implemented as a relocate-able resource on PCI (refer to *Appendix Y, Locating IPMI System Interfaces on PCI*).

There are two ACPI-based mechanisms that work together when the IPMI System Interface is implemented as a static resource, the *Service Processor Management Interface* (SPMI) Description Table and ACPI Control Methods.

C3-1 SPMI Description Table and ACPI Control Methods

The SPMI Description Table is an optional table that describes the processor-relative, translated, resources of an IPMI system interface at system boot time. The purpose of the SPMI Table is to provide a mechanism that can be used by the OSPM (an ACPI term for “OS Operating System-directed configuration and Power Management” essentially meaning an ACPI-aware OS or OS loader) very early in the boot process, e.g., before the ability to execute ACPI control methods in the OS is available.

The SPMI Description Table is similar to the SMBIOS Type 38 (IPMI Device Information) record. The main difference between the two is that the SPMI Table is identified in the ACPI Specification as a table that has the reserved signature “SPMI”. The SMBIOS Type 38 record type is from the SMBIOS specifications from the Distributed Management Task Force (http://www.dmtf.org) pre-OS working group.

The SPMI Description Table can be used to describe the location of either fixed resource or PCI implementations of the system interface. For system interfaces on PCI, the table can only describe the location of the system interface at the time that the boot process is initiated. An OS may relocate these resources. Therefore, whether or not a PCI-based

system interface remains at the SPMI addresses is OS-dependent. During normal run-time operation, software should locate the system interface directly on PCI and/or use the OS's support for PCI instead of the SPMI Table.

A management controller device may present more than one system interface for IPMI messaging to the BMC. For example, a BMC may simultaneously support the KCS and the BT interfaces. A unique SPMI Table should be provided for each of these interfaces. This allows the OSPM to select an interface that it is able to communicate and hence maximize the supportability.

Per [ACPI 2.0], unless otherwise specified, numeric values for the table and any blocks or structures are always encoded in little endian format. Signature values are stored as fixed-length strings.

Table 2-1 Service Processor Management Interface Description Table Format

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	'SPMI'. Signature for the Service Processor Management Interface Table.
Length	4	4	Length, in bytes, of the entire Service Processor Management Interface Table.
Revision	1	8	5
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID. Per ACPI specification. An OEM-supplied string that identifies the OEM.
OEM Table ID	8	16	For the Service Processor Management Interface Table, the table ID is the manufacturer model ID (assigned by the OEM identified by "OEM ID").
OEM Revision	4	24	OEM revision of Service Processor Management Interface Table for the given OEM Table ID. Per ACPI, this is "An OEM-supplied revision number. Larger numbers are assumed to be newer revisions."
Creator ID	4	28	Vendor ID of utility that created the table. For the tables containing Definition Blocks, this is the ID for the ASL Compiler.
Creator Revision	4	32	Revision of utility that created the table. For the tables containing Definition Blocks, this is the revision for the ASL Compiler.
IPMI	1	36	This field is always 01h (for backward compatibility)
Interface Type	1	37	Indicates the type of IPMI interface: *0 Reserved* *1 Keyboard Controller Style (KCS)* *2 Server Management Interface Chip (SMIC)* *3 Block Transfer (BT)* *4-255 Reserved*
Specification Revision (version)	2	38	Identifies the IPMI specification revision, in BCD format, to which the interface was designed. The first byte holds the most significant digits, while second byte holds the least significant digits of the revision, e.g. a value of 0x0150 indicates the interface is compatible with IPMI specification v1.5.
Interrupt Type	1	40	Interrupt type(s) used by the interface: Bit[7:2] Reserved (must be 0) Bit[1] I/O APIC/SAPIC interrupt (Global System Interrupt) Bit[0] SCI triggered through GPE 0 = not supported 1 = supported
GPE	1	41	The bit assignment of the SCI interrupt within the GPEx_STS register of a GPE described if the FADT that the interface triggers. (Note: This field is valid only if Bit[0] of the Interrupt Type field is set.)
Reserved	1	42	00h.

Field	Byte Length	Byte Offset	Description
PCI Device Flag	1	43	Bit 0 – PCI Device Flag. For PCI IPMI devices, this bit is set. For non-PCI devices, this bit is cleared. When this bit is cleared, the PCI Segment Group, Bus, Device and Function Number fields combined corresponds to the ACPI _UID value of the device whose _HID or _CID contains IPI0001 plug and play ID. Bit 1-7 – Reserved
Global System Interrupt	4	44	The I/O APIC or I/O SAPIC Global System Interrupt[1] used by the interface. (Note: This field is valid only if Bit[1] of the Interrupt Type field is set.)
Base Address	12	48	The base address of the interface register set described using the Generic Address Structure (GAS, See [ACPI 2.0] for the definition). The Address_Space_ID field in the GAS can only be of the value of 0 (System Memory) and 1 (System IO). All other values are not permitted.
PCI Segment Group Number	1	60	PCI Segment Group Number, if the IPMI device is a PCI device
PCI Bus Number	1	61	PCI Bus Number, if the IPMI device is a PCI device
PCI Device Number	1	62	Bit 4:0 – PCI Device Number: The PCI device number if the IPMI device is a PCI device. Bit 7:5 – Reserved
PCI Function Number	1	63	Bit 2:0 – PCI Function Number: The PCI function number if the IPMI device is a PCI device. Bit 5:3 – Reserved Bit 6 – Interrupt Flag: 0-interrupt not supported, 1-interrupt supported Bit 7 – Reserved

1. ACPI represents all interrupts as “flat” values known as global system interrupts. Therefore to support APICs or SAPICs on an ACPI-enabled system, each used APIC or SAPIC interrupt input must be mapped to the global system interrupt value used by ACPI. See Section “Global System Interrupts” in [ACPI 2.0] for a description of Global System Interrupts.

C3-1 Locating IPMI System Interfaces in ACPI Name Space

The SPMI Description Table provides a mechanism that can be used before the ability to execute ACPI control methods in the OS is available. This table is not, however, intrinsically supported in the OSPM as a way of discovering and reporting system resources. Therefore, it is recommended that non-PCI IPMI System Interfaces on the baseboards be described in the ACPI name space. This makes it possible for the OSPM to enumerate the IPMI System Interface as a device. In addition, the ACPI name space description is more flexible and friendly in hot-plug scenarios.

Note that to be ACPI compatible, the fixed resources for IPMI System Interfaces must still be accounted for in accordance with the ACPI specification. If the device is not formally described in the ACPI Name Space, its resources must be described as fixed system resources or the resources appended to some other fixed resource system device in order to ensure that the OSPM does not attempt to allocate those resources to some other device.

To formally describe the IPMI System Interface in ACPI Name Space, an IPMI device is created using the named device object. The IPMI device object can have the following elements:

Table C3-1 IPMI Device Object Control Methods

Object	Description	Support Level
_ADR	Named object that evaluates to the interface's address on its parent bus. _ADR is a standard device configuration control method defined in the ACPI Specification.	Required only for devices on a bus that has standard enumeration mechanism.
_HID	Named object that provides the interface's Plug and Play identifier. This value can be vendor specific but must set to IPI0001[2] if no _CID object is provided. _HID is a standard device configuration control method defined in the ACPI Specification.	Required
_CID	Named object that provides the interface's compatible Plug and Play identifier. This object is required and contains the value of IPI0001 if _HID contains vendor specific identifier. Otherwise, this object is optional.	
_STR	Named object that evaluates to a Unicode string that may be used by an OS to provide information to an end user describing the device. __STR is a standard device configuration control method defined in the ACPI Specification.	Required
_UID	Named object that specifies a device's unique persistent ID, or a control method that generates it. _UID is a standard device configuration control method defined in the ACPI Specification.	Required if more than one device
_CRS	Named object that returns the interface's current resource settings. System Processor Management Interfaces are considered static resources; hence only return their defined resources. The address region definition is interface type/subtype dependent. _CRS is a standard device configuration control method defined in the ACPI Specification.	Required
_STA	Object that returns the status of the device: enabled, disabled or removed, as defined in the ACPI Specification. If this method is not present, the device is assumed to be enabled.	Recommended
_IFT	Object that specifies the interface type, as defined in the SPMI Table.	Required
_SRV	Object that specifies the specification revision, as defined in the SPMI Table.	Required
_GPE	Named object that evaluates to either an integer or a package. If _GPE evaluates to an integer, the value is the bit assignment of the SCI interrupt within the GPEx_STS register of a GPE block described in the FADT that the Service Processor Management Interface will trigger. If _GPE evaluates to a package, then that package contains two elements. The first is an object reference to the GPE Block device that contains the GPE register that will be triggered by the interface. The second element is numeric (integer) that specifies the bit assignment of the SCI interrupt within the GPEx_STS register of the GPE Block device referenced by the first element in the package. (Note: This object is only provided if the interface supports a GPE.)	Required if interrupt through GPE is supported

NOTE: Normally PCI based devices are not described in ACPI name space. OSPM should use the PCI enumeration mechanism to locate IPMI interfaces. See *Appendix Y, Locating IPMI System Interfaces on PCI*.

If the IPMI interface supports interrupts, the interrupt descriptor in _CRS is used if the interrupt is supported via IO (S)APIC, while _GPE object is used if the interrupt is supported through the GPE register. Having both the interrupt descriptor in _CRS and the _GPE object in the IPMI device scope is not permitted by this specification.

If the IPMI interface does not support interrupts, neither the interrupt descriptor in the _CRS nor the _GPE object will be present.

[2] Intel has registered the IPIxxxx PNP ID with Microsoft for describing all IPMI related devices. Intel has granted the use of IPI0001 to describe the generic Service Processor Management Device as defined in this specification.

In a multi-node system where there may be more than one IPMI device in an OS domain, it is highly recommended that all IPMI devices be described in the ACPI name space with the _STA returning enabled for the active IPMI device(s).

C3-2 Example IPMI Definition ASL Code

Example ASL code that defines IPMI System Interfaces is shown below:

Example 1: SMIC Interface in I/O Space

Example ASL for describing an IO-port based SMIC system interface:

```
Device(MI0) {
        Name(_HID, EISAID("IPI0001"))
        Name(_STR, Unicode("IPMI_SMIC"))      // Optional, but recommended
                                              // for identifying IPMI system interface.
                                    // The strings "IPMI_KCS", "IPMI_SMIC",
                                              // and "IPMI_BT" are recommended for
                                              // identifying the KCS, SMIC, and BT
                                              // interfaces, respectively.

        Name(_UID, 0)          // UID for the primary IPMI system interface in the system

        // Returns the "Current Resources"
        Name(_CRS,
        ResourceTemplate() {
                IO(Decode16, 0xCA9, 0, 3)     // Ports 0xCA9, 0xCAA & 0xCAB
        }
        )

        // Returns the interface type
        Method(_IFT) {
                Return(0x02)   // IPMI SMIC
        }

        // Returns the interface specification revision
        Method(_SRV) {
                Return(0x0100) // IPMI Specification Revision 1.0
        }

        //This interface does not support interrupt

}
```

Example 2: KCS Interface in 64-bit Address Space

Example ASL for describing a memory-mapped KCS system interface, located in a 64-bit address space at address 0x80000FFFFC020CA2:

```
Device(MI0) {
        Name(_HID, EISAID("IPI0001"))
        Name(_STR, Unicode("IPMI_KCS"))       // Optional, but recommended
                                              // for identifying IPMI system interface.
                                              // The strings "IPMI_KCS", "IPMI_SMIC",
                                              // and "IPMI_BT" are recommended for
                                              // identifying the KCS, SMIC, and BT
                                              // interfaces, respectively.

        Name(_UID, 0)          // UID for the primary IPMI system interface in the system

        // Returns the "Current Resources"
        Name(_CRS,
        ResourceTemplate() {
        QWordMemory(
                ResourceConsumer,             //
                PosDecode,                    //
                MinFixed,                     //
                MaxFixed,                     //
                NonCacheable,                 //
                ReadWrite,                    //
                0xFFFFFFFFFFFFFFFF,           // _GRA, Address granularity.
                                              // E.g. All 64-bits decoded.
                0x80000FFFFC020CA2,           // _MIN, Address range minimum
                                              // (System I/F base addr.)
```

```
                0x80000FFFFC020CA4,             // _MAX, Address range max
                0x0000000000000000,             // _TRA, Translation.
                                                // 0 for non-bridge devices
                0x0000000000000002,             // _LEN, Address range length
                ,                               // Resource Source Index
                ,                               // Resource Source Name
                ,                               // A name to refer back to this resource
                ,                               // _MTP, Nothing=>AddressRangeMemory
                ,                               // _TTP, Translation. Nothing=>TypeStatic
                                        // TypeTranslation:  This resource, which is memory
                                        // on the secondary side of the bridge is I/O on the
                                        // primary side of the bridge.
                                        // TypeStatic:  This resource, which is memory on
                                        // the secondary side of the bridge is also memory
                                        // on the primary side of the bridge.
        )
        }
        )

        // Returns the interface type
        Method(_IFT) {
                Return(0x01)   // IPMI KCS
        }

        // Returns the interface specification revision
        Method(_SRV) {
                Return(0x0100) // IPMI Specification Revision 1.0
        }

        // This interface does not support interrupt

}
```

Example 3: SMIC Interface in I/O Space

Example ASL for describing a memory-mapped BT system interface using a fixed interrupt

```
Device(MI0) {
        Name(_HID, EISAID("IPI0001"))
        Name(_STR, Unicode("IPMI_BT"))          // Optional, but recommended
                                                // for identifying IPMI system interface.
                                                // The strings "IPMI_KCS", "IPMI_SMIC",
                                                // and "IPMI_BT" are recommended for
                                                // identifying the KCS, SMIC, and BT
                                                // interfaces, respectively.

        Name(_UID, 0)           // UID for the primary IPMI system interface in the system

        // Returns the "Current Resources"
        Name(_CRS,
        ResourceTemplate() {
                IO(Decode16, 0x0E4, 0, 3)       // Ports 0xE4h:E6h
                Interrupt(ResourceProducer,…){20} // GSI is 20
        }
        )

        // Returns the interface type
        Method(_IFT) {
                Return(0x03)   // IPMI BT
        }

        // Returns the interface specification revision
        Method(_SRV) {
                Return(0x0150) // IPMI Specification Revision 1.5
        }

}
```

9/26/03 Addenda, Errata, and Clarifications

E268 Typo - Table 18-9, Send Message Command

When bridging requests between IPMB and LAN, the corresponding response data is carried in the *Send Message* response. Therefore, the byte offset for the Response Data should be listed as (2:N) instead of (2) to account for the variable number of bytes of corresponding response data returned for the bridged request.

Table 18-9, Send Message Command

...

(2:N)~~(2)~~	Response Data This data will only be present when using the *Send Message* command to originate requests from IPMB or PCI Management Bus to other channels such as LAN or serial/modem. It is not present in the response to a *Send Message* command delivered via the System Interface.

E291 Addendum - Table 36-3, Sensor Type Codes, Sensor Type and Extensions for FRU Insertion / Removal

Two new sensor types to support returning info re: hot-swapping FRU's. These additions help support AdvancedTCA-type and other modular system use of IPMI.

Table 36-3, Sensor Type Codes

...

Button	14h	00h	Power Button pressed
		01h	Sleep Button pressed
		02h	Reset Button pressed
		03h	FRU latch open (Switch indicating FRU latch is in 'unlatched' position and FRU is mechanically removable)
		04h	FRU service request button (1 = pressed, service, e.g. removal/replacement, requested)

...

FRU State	2Ch	00h	FRU Not Installed
		01h	FRU Inactive (in standby or 'hot spare' state)
		02h	FRU Activation Requested
		03h	FRU Activation In Progress
		04h	FRU Active
		05h	FRU Deactivation Requested
		06h	FRU Deactivation In Progress
		07h	FRU Communication Lost

			The Event Data 2 field for this command can be used to provide the cause of the state change and the previous state: 7:4 Cause of state change 0h = Normal State Change. 1h = Change Commanded by software external to FRU. 2h = State Change due to operator changing a Handle latch. 3h = State Change due to operator pressing the hot swap push button. 4h = State Change due to FRU programmatic action. 5h = Communication Lost. 6h = Communication Lost due to local failure. 7h = State Change due to unexpected extraction. 8h = State Change due to operator intervention/update. 9h = Unable to compute IPMB address. Ah = Unexpected Deactivation. Fh = State Change, Cause Unknown. All other = reserved 3:0 Previous state offset value (return offset for same state as present state if previous state is unknown) All other = reserved.

Last Page